Doc. Ref. AO2
Appl. No. 10/925,314

(12) UK Patent Application (19) GB (11) 2 289 353 (13) A

(43) Date of A Publication 15.11.1995

(21) Application No 9408765.7

(22) Date of Filing 03.05.1994

(71) Applicant(s)
Advanced Risc Machines Limited

(Incorporated in the United Kingdom)

Fulbourn Road, Cherry Hinton, CAMBRIDGE, CB1 4JN, United Kingdom

(72) Inventor(s)
David Vivian Jaggar

(74) Agent and/or Address for Service
D Young & Co
21 New Fetter Lane, LONDON, EC4A 1DA, United Kingdom

(51) INT CL^6
G06F 9/30

(52) UK CL (Edition N)
G4A APX

(56) Documents Cited
US 4274138 A

(58) Field of Search
UK CL (Edition M) **G4A APM APX**
INT CL^5 **G06F 9/30**
ONLINE DATABASE : WPI

(54) Data processing with multiple instruction sets.

(57) A data processing system utilising two instruction sets. Both instruction sets control processing using full N-bit data pathways within a processor core 2. One instruction set is a 32-bit instruction set and the other is a 16-bit instruction set. Both instruction sets are permanently installed and have associated instruction decoding hardware 30, 36, 38.

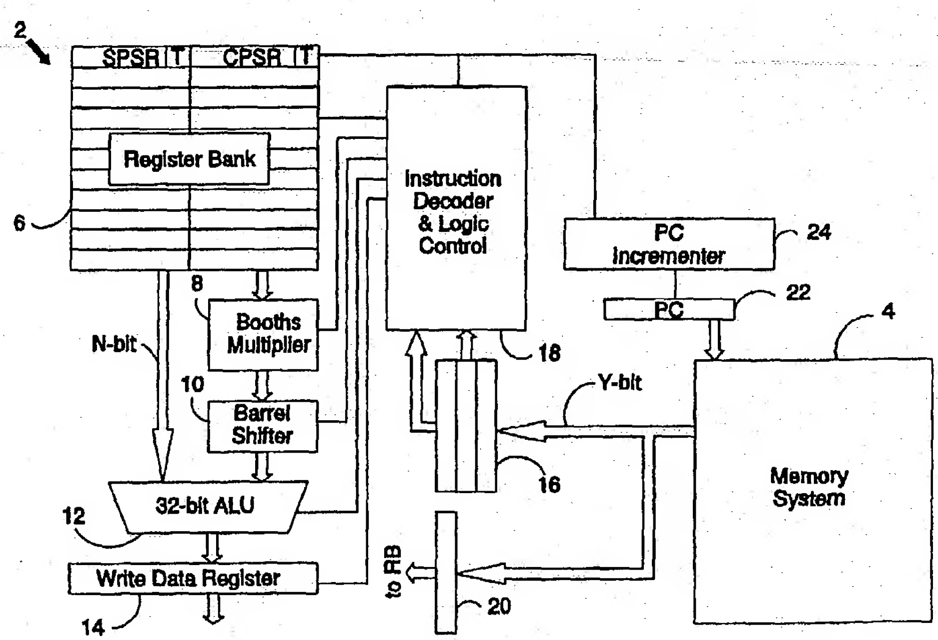


Fig.1

GB 2 289 353 A

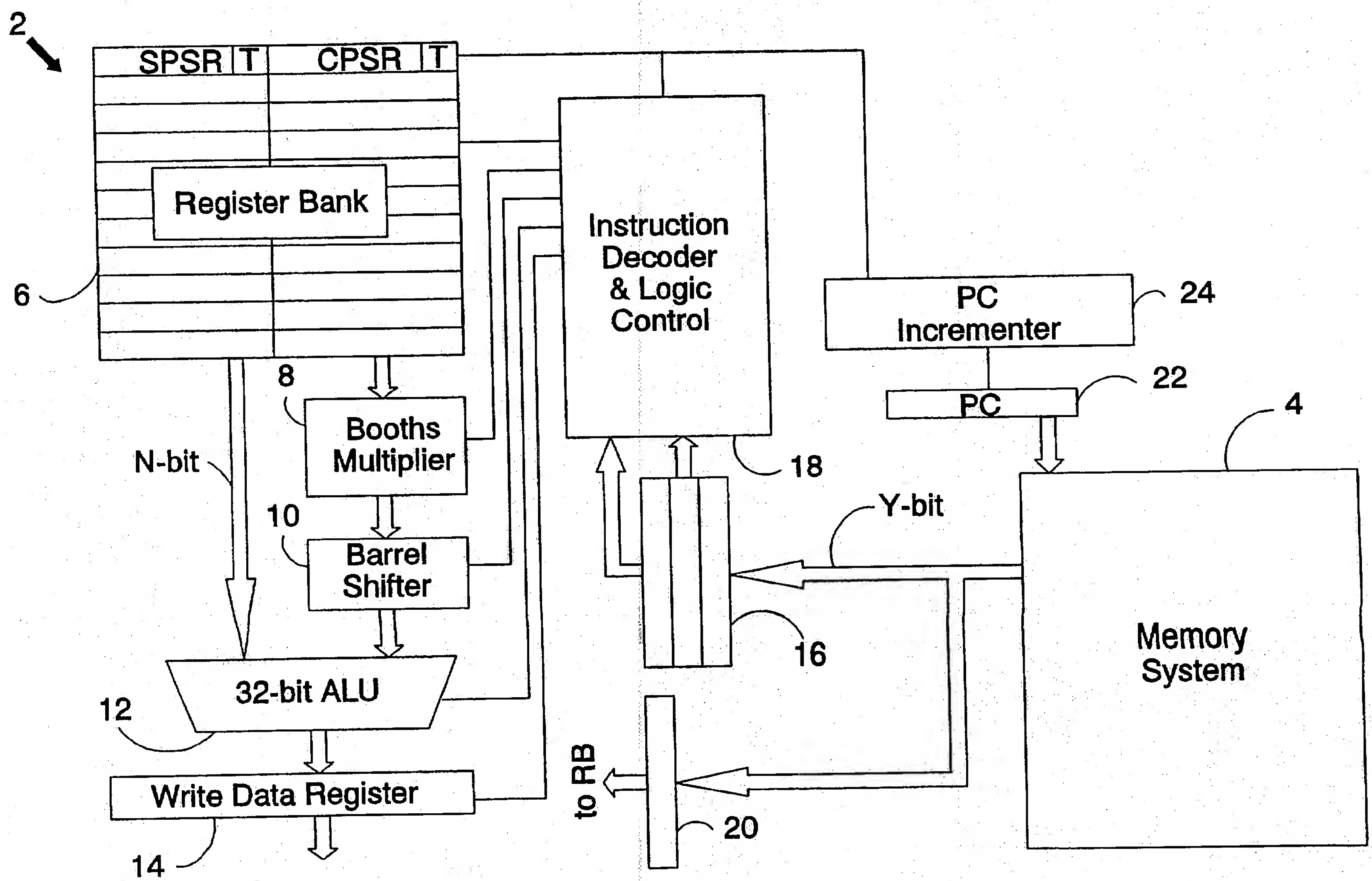

2
SPSR
T
CPSR
T
Register Bank
6
8
Booths Multiplier
10
Barrel Shifter
N-bit
32-bit ALU
12
Write Data Register
14
Instruction Decoder & Logic Control
18
16
Y-bit
to RB
20
PC Incrementer
24
PC
22
4
Memory System

Fig. 1

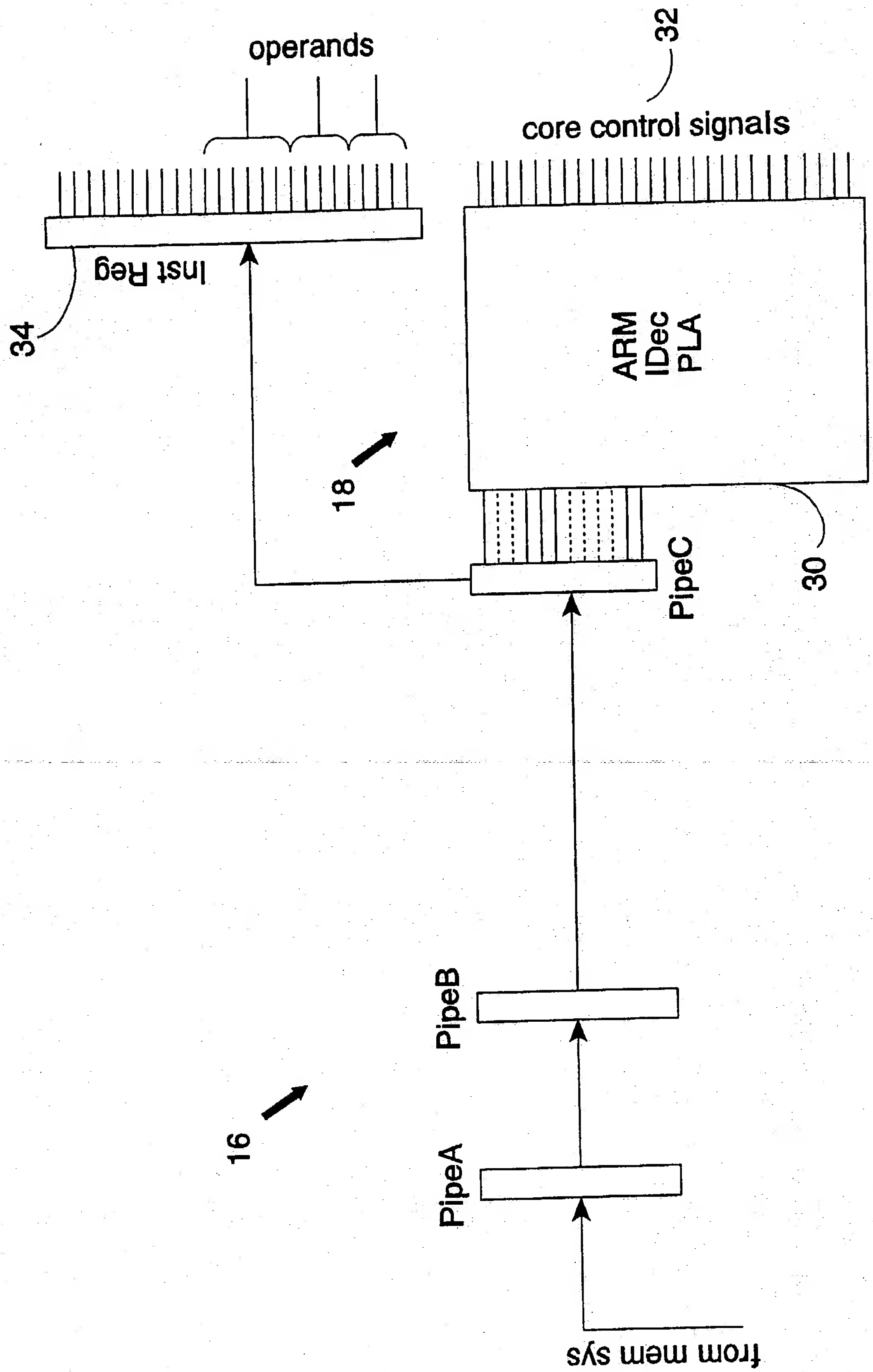

operands
32
core control signals
34
Inst Reg
ARM
IDec
PLA
18
PipeC
30
PipeB
16
PipeA
from mem sys

Fig.2

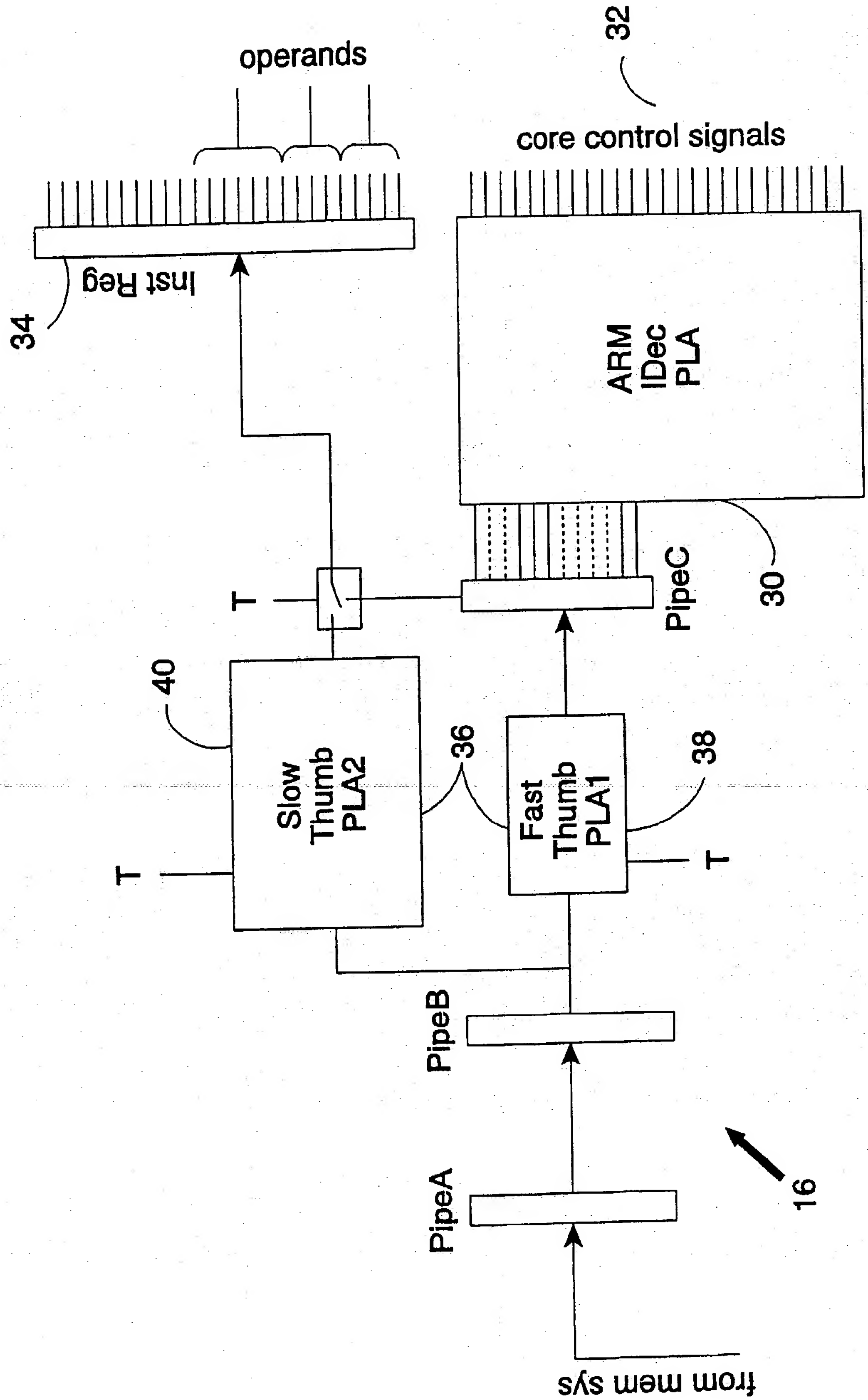
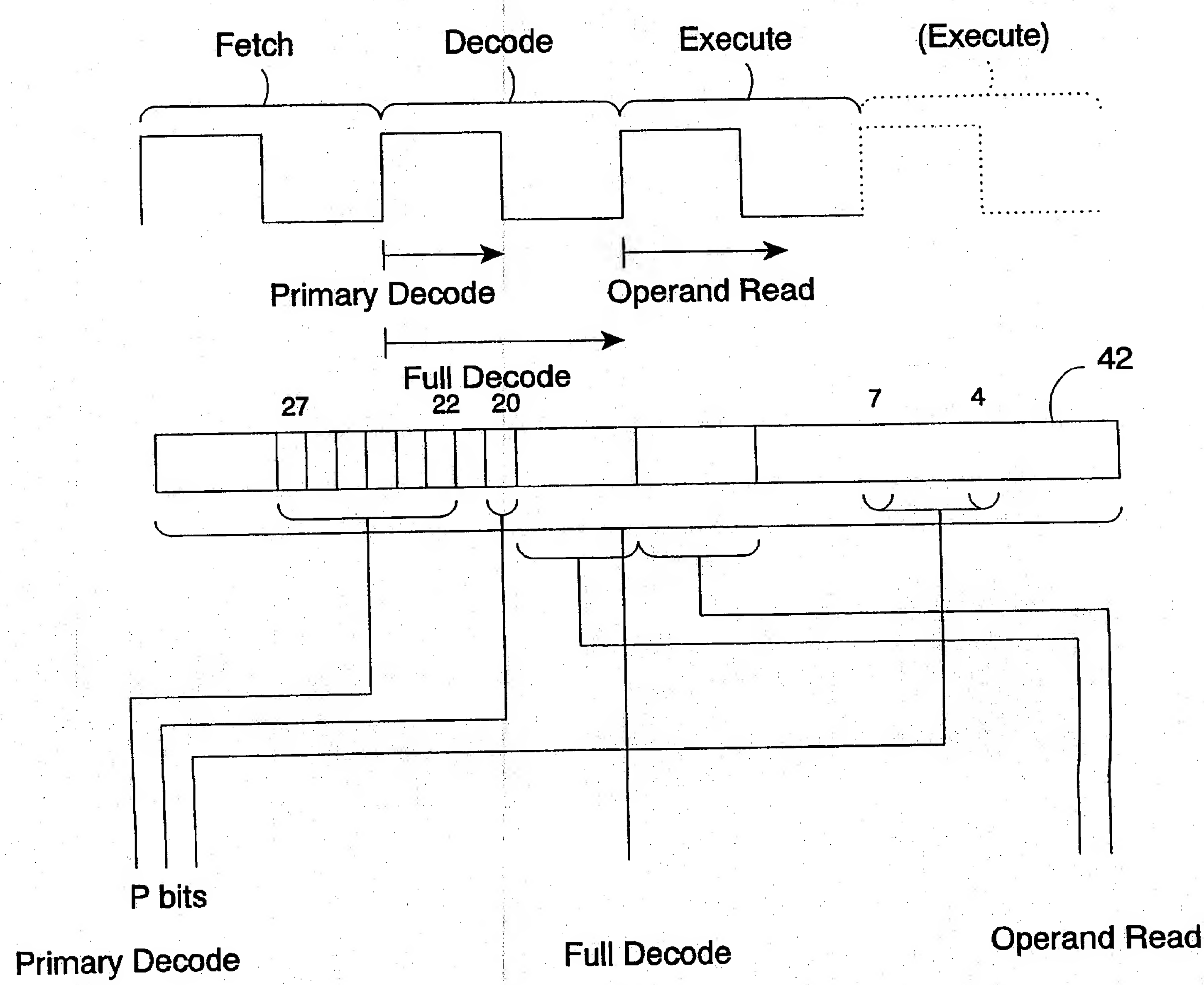
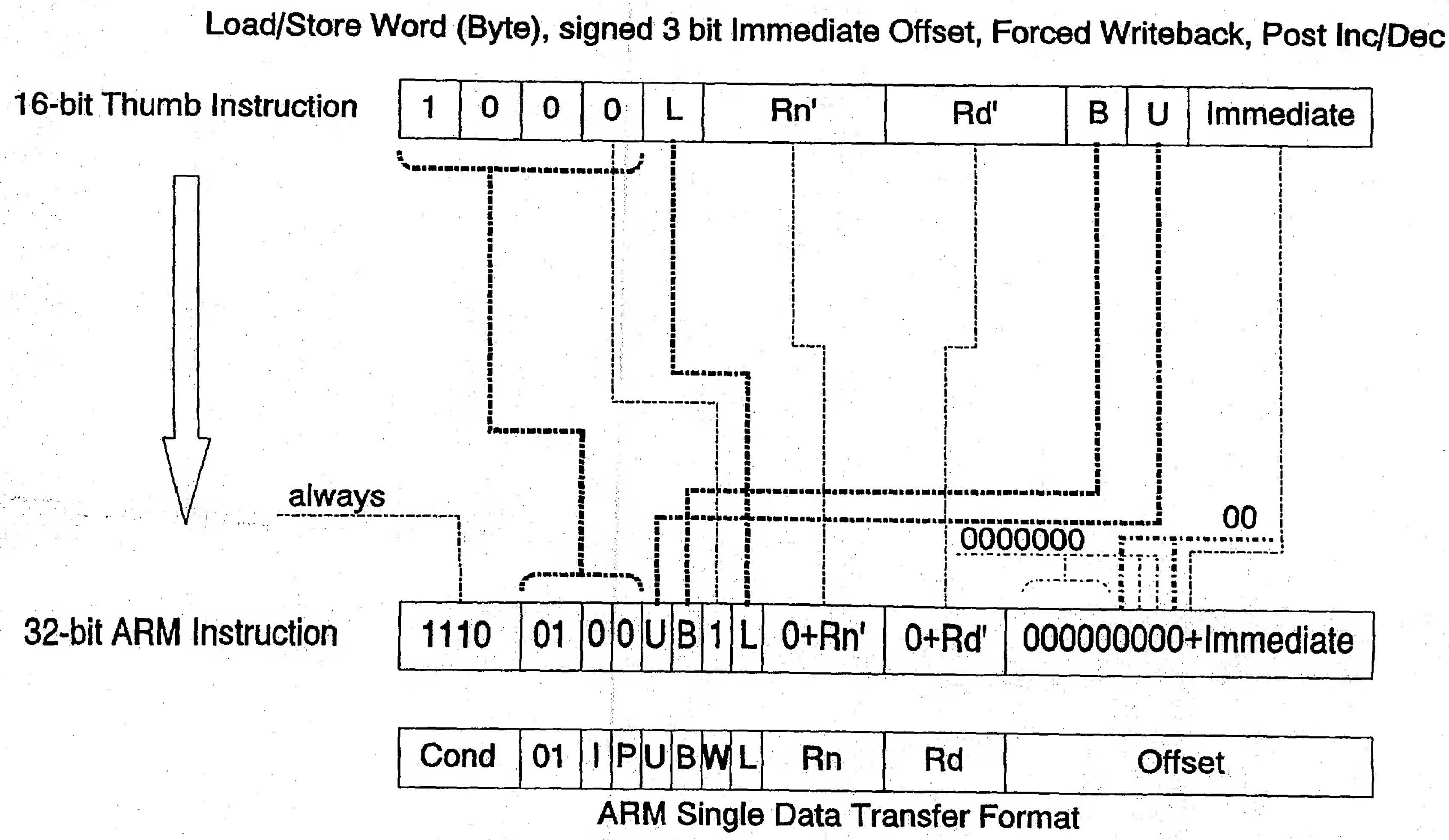

operands
32
core control signals
34
Inst Reg
ARM
IDec
PLA
T
40
30
PipeC
Slow
Thumb
PLA2
36
Fast
Thumb
PLA1
38
T
T
PipeB
PipeA
16
from mem sys

Fig.3



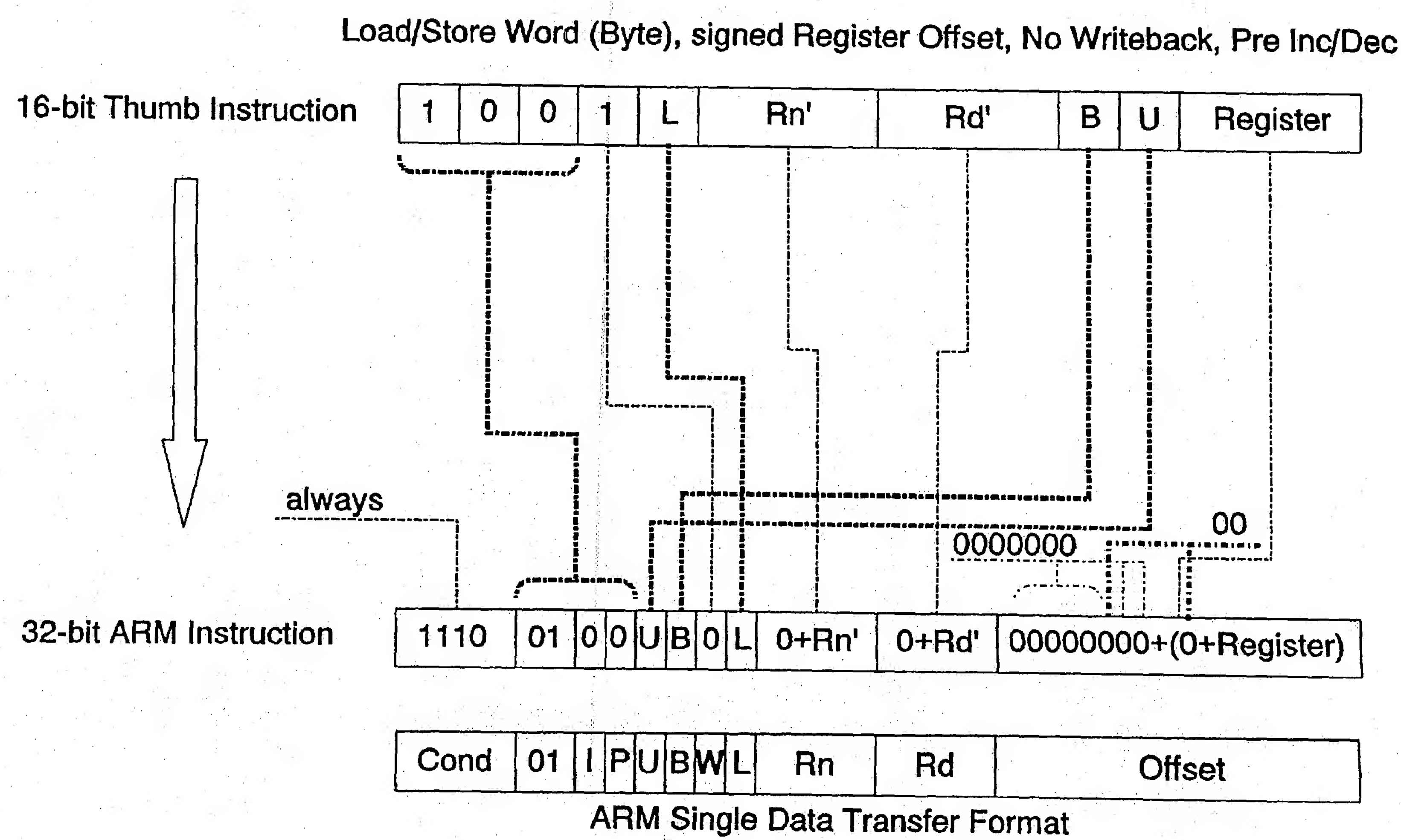
Fetch
Decode
Execute
(Execute)
Primary Decode
Operand Read
Full Decode
27
22
20
7
4
42
P bits
Primary Decode
Full Decode
Operand Read

Fig.4



Load/Store Word (Byte), signed 3 bit Immediate Offset, Forced Writeback, Post Inc/Dec
16-bit Thumb Instruction
1
0
0
0
L
Rn'
Rd'
B
U
Immediate
always
00
0000000
32-bit ARM Instruction
1110
01
0
0
U
B
1
L
0+Rn'
0+Rd'
000000000+Immediate
Cond
01
I
P
U
B
W
L
Rn
Rd
Offset
ARM Single Data Transfer Format

Fig.5



Load/Store Word (Byte), signed Register Offset, No Writeback, Pre Inc/Dec
16-bit Thumb Instruction
1
0
0
1
L
Rn'
Rd'
B
U
Register
always
00
0000000
32-bit ARM Instruction
1110
01
0
0
U
B
0
L
0+Rn'
0+Rd'
00000000+(0+Register)
Cond
01
I
P
U
B
W
L
Rn
Rd
Offset
ARM Single Data Transfer Format

Fig.6

31			28	27	26	25	24	23	22	21	20	19			16	15			12	11			8	7		5	4	3			0
Cond				00		1	OpCode				S	Rn				Rd				Operand2											
Cond				000000						A	S	Rd				Rn				Rs				1001				Rm			
Cond				00010					B	00		Rn				Rd				0000				1001				Rm			
Cond				01		I	P	U	B	W	L	Rn				Rd				Offset											
Cond				011			xxxxxxxxxxxxxxxxxxxx																				1	xxxx			
Cond				100			P	U	S	W	L	Rn				Register List															
Cond				101			L	Offset																							
Cond				110			P	U	N	W	L	Rn				CRd				CP#				Offset							
Cond				1110				CP Opc				CRn				CRd				CP#				CP			0	CRm			
Cond				1110				CP Opc			L	CRn				Rd				CP#				CP			1	CRm			
Cond				1111				ignored by processor																							

Fig.7

15	14	13	12	11	10		8	7		5	4	3	2		0
0	0	0	00		Rd			Rs			0	OP	Immediate		
0	0	0	00		Rd			Rs			1	OP	Register		
0	0	0	OP=1-3		Rd			Immediate							
0	0	0	OP=0-2		Rd			Rs			Immediate				
0	0	1	1	1	OP1			Rd/Rs1			OP2		Rs2		
0	1	0	OP		Rd/Rs			Immediate							
0	1	0	1	1	Rd			PC Relative Offset							
0	1	1	B	L	Rd/Rs			SP Relative Offset							
1	0	0	0	L	Rn			Rd/Rs			B	U	Immediate		
1	0	0	1	L	Rn			Rd/Rs			B	U	Register		
1	0	1	B	L	Rn			Rd/Rs			Immediate				
1	1	0	0	L	Rn			Register List							
1	1	0	1	SP	Rd			Effective Address Offset							
1	1	1	0	Cond				8 Bit Branch Offset							
1	1	1	1	0	Long Branch and Link Segment										
1	1	1	1	1	Long Branch and Link Offset										

Fig.8

Thumb Register	ARM Register
General Register 0	General Register 0
General Register 1	General Register 1
General Register 2	General Register 2
General Register 3	General Register 3
General Register 4	General Register 4
General Register 5	General Register 5
General Register 6	General Register 6
General Register 7	General Register 7
	General Register 8
	General Register 9
Stack Limit	General Register 10
	General Register 11
	General Register 12
Stack Pointer	Stack Pointer (R13)
Link Register	Link Register (R14)
Program Counter	Program Counter (R15)
CPSR	CPSR
SPSR	SPSR

Fig.9

DATA PROCESSING WITH MULTIPLE INSTRUCTION SETS

This invention relates to the field of data processing. More particularly, this invention relates to data processing utilizing multiple sets of program instruction words.

Data processing systems utilize a processor core operating under control of program instruction words, which when decoded serve to generate control signals to control the different elements within the processor core to perform the necessary functions to achieve the processing specified in the program instruction word.

A typical processor core will have data pathways of a given bit width that limit the length of the data words that can be manipulated in response to a given instruction. The trend in the field of data processing has been for a steady increase in these data pathway widths, e.g. a gradual move from 8-bit architectures to 16-bit, 32-bit and 64-bit architectures. At the same time as this increase in data pathway width, the instruction sets have increased in the number of instructions possible (in both the CISC and RISC philosophies) and the bit length of those instructions. As an example, there has been a move from the use of 16-bit architectures with 16-bit instruction sets to the use of 32-bit architectures with 32-bit instruction sets.

A problem with migration towards increased architecture widths is the desire to maintain backward compatibility with program software written for preceding generations of machines. One way of addressing this has been to provide the new system with a compatibility mode. For example, the VAX11 computers of Digital Equipment Corporation have a compatibility mode that enables them to decode the instructions for the earlier PDP11 computers. Whilst this allows the earlier program software to be used, such use is not taking full advantage of the increased capabilities of the new processing system upon which it is running, e.g. perhaps only multiple stage 16-bit arithmetic is being used when the system in fact has the hardware to support 32-bit arithmetic.

Another problem associated with such changes in architecture width is that the size of computer programs using the new increased bit

width instruction sets tends to increase (a 32-bit program instruction word occupies twice the storage space of a 16-bit program instruction word). Whilst this increase in size is to some extent offset by a single instruction being made to specify an operation that might previously have needed more than one of the shorter instructions, the tend is still for increased program size.

An approach to dealing with this problem is to allow a user to effectively specify their own instruction set. The IBM370 computers made by International Business Machines Corporation incorporate a writable control store using which a user may set up their own individual instruction set mapping instruction program words to desired actions by the different portions of the processor core. Whilst this approach gives good flexibility, it is difficult to produces high speed operation and the writable control store occupies a disadvantageously large area of an integrated circuit. Furthermore, the design of an efficient bespoke instruction set is a burdensome task for a user.

It is also known to provide systems in which a single instruction set has program instruction words of differing lengths. An example of this approach is the 6502 microprocessor produced by MOS Technology. This processor uses 8-bit operation codes that are followed by a variable number of operand bytes. The operation code has first to be decoded before the operands can be identified and the instruction effected. This requires multiple memory fetches and represents a significant constraint on system performance compared with program instructions words (i.e. operation code and any operands) of a constant known length.

Viewed from one aspect the invention provides apparatus for processing data, said apparatus comprising:

a processor core having N-bit data pathways and being responsive to a plurality of core control signals;

first decoding means for decoding X-bit program instruction words from a first permanent instruction set to generate said core control signals to trigger processing utilizing said N-bit data pathways;

second decoding means for decoding Y-bit program instruction words from a second permanent instruction set to generate said core

control signals to trigger processing utilizing said N-bit data pathways, Y being less than X; and

an instruction set switch for selecting either a first processing mode using said first decoding means upon received program instruction words or a second processing mode using said second decoding means upon received program instruction words.

The invention recognises that in a system having a wide standard X-bit instruction set and N-bit data pathways (e.g. a 32-bit instruction set operating on 32-bit data pathways), the full capabilities of the X-bit instruction set are often not used in normal programming. An example of this would be a 32-bit branch instruction. This branch instruction might have a 32 megabyte range that would only very occasionally be used. Thus, in most cases the branch would only be for a few instructions and most of the bits within the 32-bit instruction would be carrying no information. Many programs written using the 32-bit instruction set would have a low code density and utilize more program storage space than necessary.

The invention addresses this problem by providing a separate permanent Y-bit instruction set, where Y is less than X, that still operates on the full N-bit data pathways. Thus, the performance of the N-bit data pathways is utilized whilst code density is increased for those applications not requiring the sophistication of the X-bit instruction set.

There is a synergy in the provision of the two permanent instruction sets. The user is allowed the flexibility to alter the instruction set they are using to suit the circumstances of the program, with both instruction sets being efficiently implemented by the manufacturer (critical in high performance systems such as RISC processors where relative timings are critical) and without sacrificing the use of the N-bit data pathways.

Another advantage of this arrangement is that since fewer bytes of program code will be run per unit time when operating with the Y-bit instruction set, less stringent demands are place upon the data transfer capabilities of the memory systems storing the program code. This reduces complexity and cost.

The invention also moves in the opposite direction to the usual trend in the field. The trend is that with each new generation of processors, more instructions are added to the instructions sets with the instruction sets becoming wider to accommodate this. In contrast, the invention starts with a wide sophisticated instruction set and then adds a further narrower instruction set (with less space for large numbers of instructions) for use in situations where the full scope of the wide instruction set is not required.

It will be appreciated that the first instruction set and the second instruction set may be completely dependent. However, in preferred embodiments of the invention said second instruction set provides a subset of operations provided by said first instruction set.

Providing that the second instruction set is a sub-set of the first instruction set enables more efficient operation since the hardware elements of the processor core may be set out more readily to suit both instruction sets.

When an instruction set of program instruction words of an increased bit length has been added to an existing program instruction set, it is possible to ensure that the program instruction words from the two instruction sets are orthogonal. However, the instruction set switch allows this constraint to be avoided and permits systems in which said second instruction set is non-orthogonal to said first instruction set.

The freedom to use non-orthogonal instruction sets eases the task of the system designer and enables other aspects of the instruction set design to be more effectively handled.

The instruction set switch could be a hardware type switch set by some manual intervention. However, in preferred embodiments of the invention said instruction set switch comprises means responsive to an instruction set flag, said instruction set flag being setable under user program control.

Enabling the instruction set switch to be used to switch between the first instruction set and the second instruction set under software control is a considerable advantage. For example, a programmer may utilise the second instruction set with its Y-bit program instruction

words for reasons of increased code density for the majority of a program and temporarily switch to the first instruction set with its X-bit program instruction words for those small portions of the program requiring the increased power and sophistication of the first instruction set.

The support of two independent instruction sets may introduce additional complication into the system. In preferred embodiments of the invention said processor core comprises a program status register for storing currently applicable processing status data and a saved program status register, said saved program status register being utilized to store processing status data associated with a main program when a program exception occurs causing execution of an exception handling program, said instruction set flag being part of said processing status data.

Providing the instruction set flag as part of the programming status data ensures that it is saved when an exception occurs. In this way, a single exception handler can handle exceptions from both processing modes and can be allowed access to the saved instruction set flag within the saved program status register should this be significant in handling the exception. Furthermore, the exception handler can be made to use either instruction set to improve either its speed or code density as the design constraints require.

In order to deal with the differing bit lengths of the different instruction sets, preferred embodiments of the invention provide that said processor core comprises a program counter register and a program counter incrementer for incrementing a program counter value stored within said program counter register to point to a next program instruction word, said program counter incrementer applying a different increment step in said first processing mode than in said second processing mode.

It will be appreciated that the shorter program instruction words of the second instruction set cannot contain as much information as those of the first instruction set. In order to accommodate this it is preferred that the spaces saved within the second instruction set by

reducing the operand range that may be specified within a program instruction word.

In preferred embodiments of the invention said processor core is coupled to a memory system by a Y-bit data bus, such that program instruction words from said second instruction set require a single fetch cycle and program instruction words from said first instruction set require a plurality of fetch cycles.

The use of a Y-bit data bus and memory system allows a less expensive total system to be built whilst still enabling a single fetch cycle for each program instruction word for at least the second instruction set.

The first decoding means and the second decoding means may be completely separate. However, in preferred embodiments of the invention said second decoding means reuses at least a part of said first decoding means.

The re-use of at least part of the first decoding means by the second decoding means reduces the overall circuit area. Furthermore, since the first instruction set is generally less complicated then the second instruction set and is driving the same processor core, there will be a considerable amount of the second decoding means that it is possible to re-use.

Viewed from another aspect the invention provides a method of processing data, said method comprising the steps of:

selecting either a first processing mode or a second processing mode for a processor core having N-bit data pathways and being responsive to a plurality of core control signals;

in said first processing mode, decoding X-bit program instruction words from a first permanent instruction set to generate said core control signals to trigger processing utilizing said N-bit data pathways; and

in said second processing mode, decoding Y-bit program instruction words from a second permanent instruction set to generate said core control signals to trigger processing utilizing said N-bit data pathways, Y being less than X.

An embodiment of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 schematically illustrates a data processing apparatus incorporating a processor core and a memory system;

Figure 2 schematically illustrates an instruction and instruction decoder for a system having a single instruction set;

Figure 3 illustrates an instruction pipeline and instruction decoders for use in a system having two instruction sets;

Figure 4 illustrates the decoding of an X-bit program instruction word;

Figures 5 and 6 illustrate the mapping of Y-bit program instruction words to X-bit program instruction words;

Figure 7 illustrates an X-bit instruction set;

Figure 8 illustrates a Y-bit instruction set; and

Figure 9 illustrates the processing registers available to the first instruction set and the second instruction set.

Figure 1 illustrates a data processing system (that is formed as part of an integrated circuit) comprising a processor core 2 coupled to a Y-bit memory system 4. In this case, Y is equal to 16.

The processor core 2 includes a register bank 6, a Booths multiplier 8, a barrel shifter 10, a 32-bit arithmetic logic unit 12 and a write data register 14. Interposed between the processor core 2 and the memory system 4 is an instruction pipeline 16, an instruction decoder 18 and a read data register 20. A program counter register 22, which is part of the processor core 2, is shown addressing the memory system 4. A program counter incrementer 24 serves to increment the program counter value within the program counter register 22 as each instruction is executed and a new instruction must be fetched for the instruction pipeline 16.

The processor core 2 incorporates N-bit data pathways (in this case 32-bit data pathways) between the various functional units. In operation, instructions within the instruction pipeline 16 are decoded by the instruction decoder 18 which produces various core control signals that are passed to the different functional elements within the processor core 2. In response to these core control signals, the different portions of the processor core conduct 32-bit processing

operations, such as 32-bit multiplication, 32-bit addition and 32-bit logical operations.

The register bank 6 includes a current programming status register 26 and a saved programming status register 28. The current programming status register 26 holds various condition and status flags for the processor core 2. These flags may include processing mode flags (e.g. system mode, user mode, memory abort mode etc.) as well as flags indicating the occurrence of zero results in arithmetic operations, carries and the like. The saved programming status register 28 (which may be one of a banked plurality of such saved programming status registers) is used to temporarily store the contents of the current programming status register 26 if an exception occurs that triggers a processing mode switch. In this way, exception handling can be made faster and more efficient.

Included within the current programming status register 26 is an instruction set flag T. This instruction set flag is supplied to the instruction decoder 18 and the program counter incrementer 24. When this instruction set flag T is set, the system operates with the instructions of the second instruction set (i.e. Y-bit program instruction words, in this case 16-bit program instruction words). The instruction set flag T controls the program counter incrementer 24 to adopt a smaller increment step when operated with the second instruction set. This is consistent with the program instruction words of the second instruction set being smaller and so more closely spaced within the memory locations of the memory system 4.

As previously mentioned, the memory system 4 is a 16-bit memory system connected via 16-bit data buses to the read data register 20 and the instruction pipeline 16. Such 16-bit memory systems are simpler and inexpensive relative to higher performance 32-bit memory systems. Using such a 16-bit memory system, 16-bit program instruction words can be fetched in a single cycle. However, if 32-bit instructions from the second instruction set are to be used (as indicated by the instruction set flag T), then two instruction fetches are required to recover a single 32-bit instruction for the instruction pipeline 16.

Once the required program instruction words have been recovered from the memory system 4, they are decoded by the instruction decoder 18 and initiate 32-bit processing within the processor core 2

irrespective of whether the instructions are 16-bit instructions or 32-bit instructions.

The instruction decoder 18 is illustrated in Figure 1 as a single block. However, in order to deal with more than one instruction set, the instruction decoder 18 has a more complicated structure as will be discussed in relation to Figures 2 and 3.

Figure 2 illustrates the instruction pipeline 16 and an instruction decoder 18 for coping with a single instruction set. In this case, the instruction decoder 18 includes only a first decoding means 30 that is operative to decode 32-bit instructions. This decoding means 30 decodes the first instruction set (the ARM instruction set) utilising a programmable logic array (PLA) to produce a plurality of core control signals 32 that are fed to the processor core 2. The program instruction word which is currently decoded (i.e. yields the current the core control signals 32) is also held within an instruction register 34. Functional elements within the processor core 2 (e.g. the Booths multiplier 8 or the register bank 6) read operands needed for their processing operation directly from this instruction register 34.

A feature of the operation of such an arrangement is that the first decoding means 30 requires certain of its inputs (the P bits shown as solid lines emerging from the PipeC pipeline stage) early in the clock cycle in which the first decoding means operates. This is to ensure that the core control signals 32 are generated in time to drive the necessary elements within the processor core 2. The first decoding means 30 is a relatively large and slow programmable logic array structure and so such timing considerations are important.

The design of such programmable logic array structures to perform instruction decoding is conventional within the art. A set of inputs are defined together with the desired outputs to be generated from those inputs. Commercially available software is then used to devise a PLA structure that will generate the specified set of outputs from the specified set of inputs.

Figure 3 illustrates the system of Figure 2 modified to deal with decoding a first instruction set and a second instruction set. When the first instruction set is selected by the instruction set flag T, then the system operates as described in relation to Figure 2. When

the instruction set flag T indicates that the instructions in the instruction pipeline 16 are from the second instruction set, a second decoding means 36 becomes active.

This second decoding means decodes the 16-bit instructions (the Thumb instructions) utilising a fast PLA 38 and a parallel slow PLA 40. The fast PLA 38 serves to map a subset (Q bits) of the bits of the 16-bit Thumb instructions to the P bits of the corresponding 32-bit ARM instructions that are required to drive the first decoding means 30. Since a relatively small number of bits are required to undergo this mapping, the fast PLA 38 can be relatively shallow and so operate quickly enough to allow the first decoding means sufficient time to generate the core control signals 32 in response to the contents of PipeC. The fast PLA 38 can be considered to act to "fake" the critical bits of a corresponding 32-bit instruction for the first decoding means without spending any unnecessary time mapping the full instruction.

However, the full 32-bit instruction is still required by the processor core 2 if it is to be able to operate without radical alterations and significant additional circuit elements. With the time critical mapping having been taken care of by the fast PLA 38, the slow PLA 40 connected in parallel serves to map the 16-bit instruction to the corresponding 32-bit instruction and place this into the instruction register 34. This more complicated mapping may take place over the full time it takes the fast PLA 38 and the first decoding means 30 to operate. The important factor is that the 32-bit instruction should be present within the instruction register 34 in sufficient time for any operands to be read therefrom in response to the core control signals 32 acting upon the processor core 2.

It will be appreciated that the overall action of the system of Figure 3 when decoding the second instruction set is to translate 16-bit instructions from the second instruction set to 32-bit instructions from the first instruction set as they progress along the instruction pipeline 16. This is rendered a practical possibility by making the second instruction set a subset of a first instruction set so as to ensure that there is a one to one mapping of instructions from the second instructions set into instructions within the first instruction set.

The provision of the instruction set flag T enables the second instruction set to be non-orthogonal to the first instruction set. This is particularly useful in circumstances where the first instruction set is an existing instruction set without any free bits that could be used to enable an orthogonal further instruction set to be detected and decoded.

Figure 4 illustrates the decoding of a 32-bit instruction. At the top of Figure 4 successive processing clock cycles are illustrated in which a fetch operation, a decode operation and finally an execute operation performed. If the particular instruction so requires (e.g. a multiply instruction), then one or more additional execute cycles may be added.

A 32-bit instruction 42 is composed of a plurality of different fields. The boundaries between these fields will differ for differing instructions as will be shown later in Figure 7.

Some of the bits within the instruction 42 require decoding within a primary decode phase. These P bits are bits 4, 7, 20 and 22 to 27. These are the bits that are required by the first decoding means 30 and that must be "faked" by the fast PLA 38. These bits must be applied to the first decoding means and decoded thereby to generate appropriate core control signals 32 by the end of the first part of the decode cycle. Decoding of the full instruction can, if necessary, take as long as the end of decode cycle. At the end of the decode cycle, operands within the instruction are read from the instruction register 34 by the processor 2 during the execute cycle. These operands may be register specifiers, offsets or other variables.

Figure 5 shows the mapping of an example of 16-bit instruction to a 32-bit instruction. The thick lines originate from the Q bits within the 16-bit instruction that require mapping into the P bits within the 32-bit instruction so that they may be applied to the first decoding means 30. It will be seen that the majority of these bits are either copied straight across or involve a simple mapping. The operands Rn', Rd and Immediate within the 16-bit instruction require padding at their most significant end with zeros to fill the 32-bit instruction. This padding is needed as a result of the 32-bit instruction operands having a greater range than the 16-bit instruction operands.

It will be seen from the generalised form of the 32-bit instruction given at the bottom of Figure 5, that the 32-bit instruction allows considerably more flexibility than the subset of that instruction that is represented by the 16-bit instruction. For example, the 32-bit instructions are preceded by condition codes Cond that renders the instruction conditionally executable. In contrast, the 16-bit instructions do not carry any condition codes in themselves and the condition codes of the 32-bit instructions to which they are mapped are set to a value of "1110" that is equivalent to the conditional execution state "always".

Figure 6 illustrates another such instruction mapping. The 16-bit instruction in this case is a different type of Load/Store instruction to that illustrated in Figure 5. However, this instruction is still a subset of the single data transfer instruction of the 32-bit instruction set.

Figure 7 schematically illustrates the formats of the eleven different types of instruction for the 32-bit instruction set. These instructions are in turn:

1. Data processing PSR transfer;
2. Multiply;
3. Single data swap;
4. Single data transfer;
5. Undefined;
6. Block data transfer;
7. Branch;
8. Co-processor data transfer;
9. Co-processor data operation; and
10. Co-processor register transfer.
11. Software interrupt.

A full description of this instruction set may be found in the Data Sheet of the ARM6 processor produced by Advanced RISC Machines Limited. The instruction highlighted within Figure 7 is that illustrated in Figures 5 and 6.

Figure 8 illustrates the 16-bit instruction set that is provided in addition to the 32-bit instruction set. The instructions

highlighted within this instruction set are those illustrated in Figures 5 and 6 respectively. The instructions within this 16-bit instruction set have been chosen such that they may all be mapped to a single 32-bit instruction and so form a subset of the 32-bit instruction set.

Passing in turn between each of the instructions in this instruction set, the formats specify the following:

Format 1: Op = 0,1. Both ops set the condition code flags.
0: ADD Rd, Rs, #Immediate3
1: SUB Rd, Rs, #Immediate3

Format 2: Op = 0,1. Both ops set the condition code flags.
0: ADD Rd, Rm, Rn
1: SUB Rd, Rm, Rn

Format 3: 3 opcodes. Used to build large immediates.
1 = ADD Rd, Rd, #Immediate 8<<8
2 = ADD Rd, Rd, #Immediate 8<<16
3 = ADD Rd, Rd, #Immediate 8<<24

Format 4: Op gives 3 opcodes, all operations are MOVS Rd, Rs SHIFT #Immediate5, where SHIFT is
0 is LSL
1 is LSR
2 is ASR
Shifts by zero as defined on ARM.

Format 5: Op1*8+Op2 gives 32 ALU opcodes, Rd = Rd op Rn. All operations set the condition code flags.
The operations are
AND, OR, EOR, BIC (AND NOT), NEGATE, CMP, CMN, MUL
TST, TEQ, MOV, MVN(NOT), LSL, LSR, ASR, ROR
Missing ADC, SBC, MULL
Shifts by zero and greater than 31 as defined on ARM

8 special opcodes, LO specifies Reg 0-7, HI specifies a register 8-15
SPECIAL is CPSR or SPSR
MOV HI, LO (move hidden register to visible register)
MOV LO, HI (move visible register to hidden register)
MOV HI, HI (eg procedure return)
MOVS HI, HI (eg exception return)
MOVS HI, LO (eg interrupt return, could be SUBS, HI, HI, #4)
MOV SPECIAL, LO (MSR)
MOV LO, SPECIAL (MRS)
CMP HI, HI (stack limit check)

8 free opcodes

Format 6: Op gives 4 opcodes. All operations set the condition code flags
0: MOV Rd,#Immediate 8
1: CMP Rs,#Immediate 8
2: ADD Rd, Rd,#Immediate 8
It is possible to trade ADD for ADD Rd, Rs,#Immediate5

Format 7: Loads a word PC + Offset (256 words, 1024 bytes). Note the offset must be word aligned.
LDR Rd,[PC,#+1024]
This instruction is used to access the next literal pool, to load constants, addresses etc.

Format 8: Load and Store Word from SP (r7) + 256 words (1024 bytes)
Load and Store Byte from SP(r7) + 256 bytes
LRD Rd,[SP,#+1024)
LDRB Rd,[SP,#+256]
These instructions are for stack and frame access.

Format 9: Load and Store Word (or Byte), signed 3 bit Immediate Offset (Post Inc/Dec), Forced Writeback
L is Load/Store, U is Up/Down (add/subtract offset), B is Byte/Word
LDR {B} Rd, [Rb],#+/-Offset3
STR {B} Rd, [Rb],#+/-Offset3
These instructions are intended for array access
The offset encodes 0 - 7 for bytes and 0, 4 - 28 for words

Format 10: Load and Store Word (or Byte) with signed Register Offset (Pre Inc/Dec), No writeback
L is Load/Store, U is Up/Down (add/subtract offset), B is Byte/Word
```
LDR     Rd,[Rb, +/-Ro, LSL#2]
STR     Rd,[Rb, +/-Ro, LSL#2]
LDRB    Rd,[Rb, +/-Ro]
STRB    Rd,[Rb, +/-Ro]
```
These instructions are intended for base + offset pointer access, and combined with the 8-bit MOV, ADD, SUB give fairly quick immediate offset access.

Format 11: Load and Store Word (or Byte) with signed 5 bit Immediate Offset (Pre Inc/Dec), No Writeback
L is Load/Store B is Byte/Word
LDR{B} Rd, [Rb,#+Offset5]
STR{B} Rd, [Rb,#+Offset5]
These instructions are intended for structure access
The offset encodes 0 - 31 for bytes and 0, 4 - 124 for words

Format 12: Load and Store Multiple (Forced Writeback)
LDMIA Rb!, {Rlist}

STMIA Rb!, {Rlist}
Rlist specify registers r0-r7
A sub-class of these instructions are a pair of subroutine call and return instructions.
For LDM if r7 is the base and bit 7 is set in rlist, the PC is loaded
For STM if r7 is the base and bit 7 is set in rlist, the LR is stored
If r7 is used as the base register, sp is used instead
In both cases a Full Descending Stack is implemented ie LDM is like ARM's LDMFD, STM is like ARM's STMFD
So for block copy, use r7 as the end pointer
If r7 is not the base, LDM and STM is like ARMs LDMIA, STMIA

Format 13: Load address. This instruction adds an 8 bit unsigned constant to either the PC or the stack pointer and stores the results in the destination register.
ADD Rd, sp, + 256 bytes
ADD Rd, pc, + 256 words (1024 bytes)
The SP bit indicates if the SP or the PC is the source. If SP is the source, and r7 is specified as the destination register, SP is used as the destination register.

Format 14: Conditional branch, +/- 128 bytes, where cond defines the condition code (as on ARM) cond = 15 encodes as SWI (only 256, should be plenty).

Format 15: Sets bits 22:12 of a long branch and link. MOV lr, #offset << 12.

Format 16: Performs a long branch and link. Operation is SUB newlr, pc, #4; ORR pc, oldlr, #offset <<1. newlr and oldlr mean the lr register before and after the operation.

As previously mentioned, the 16-bit instruction set has reduced operand ranges compared to the 32-bit instruction set. Commensurate with this, the 16-bit instruction set uses a subset of the registers 6 (see Figure 1) that are provided for the full 32-bit instruction set. Figure 9 illustrates the subset of registers that are used by the 16-bit instruction set.

CLAIMS

1. Apparatus for processing data, said apparatus comprising:

a processor core having N-bit data pathways and being responsive to a plurality of core control signals;

first decoding means for decoding X-bit program instruction words from a first permanent instruction set to generate said core control signals to trigger processing utilizing said N-bit data pathways;

second decoding means for decoding Y-bit program instruction words from a second permanent instruction set to generate said core control signals to trigger processing utilizing said N-bit data pathways, Y being less than X; and

an instruction set switch for selecting either a first processing mode using said first decoding means upon received program instruction words or a second processing mode using said second decoding means upon received program instruction words.

2. Apparatus as claimed in claim 1, wherein said second instruction set provides a subset of operations provided by said first instruction set.

3. Apparatus as claimed in any one of claims 1 and 2, wherein said second instruction set is non-orthogonal to said first instruction set.

4. Apparatus as claimed in any one of claims 1, 2 and 3, wherein said instruction set switch comprises means responsive to an instruction set flag, said instruction set flag being setable under user program control.

5. Apparatus as claimed in claim 4, wherein said processor core comprises a program status register for storing currently applicable processing status data and a saved program status register, said saved program status register being utilized to store processing status data associated with a main program when a program exception occurs causing execution of an exception handling program, said instruction set flag being part of said processing status data.

6. Apparatus as claimed in any one of the preceding claims, wherein said processor core comprises a program counter register and a program counter incrementer for incrementing a program counter value stored within said program counter register to point to a next program instruction word, said program counter incrementer applying a different increment step in said first processing mode than in said second processing mode.

7. Apparatus as claimed in any one of the preceding claims, wherein at least one program instruction word within said second instruction set has a reduced operand range compared to a corresponding program instruction word within said first instruction set.

8. Apparatus as claimed in any one of the preceding claims, wherein said processor core is coupled to a memory system by a Y-bit data bus, such that program instruction words from said second instruction set require a single fetch cycle and program instruction words from said first instruction set require a plurality of fetch cycles.

9. Apparatus as claimed in any one of the preceding claims, wherein said second decoding means reuses at least a part of said first decoding means.

10. Apparatus as claimed in any one of the preceding claims, wherein said apparatus is an integrated circuit.

11. A method of processing data, said method comprising the steps of:

selecting either a first processing mode or a second processing mode for a processor core having N-bit data pathways and being responsive to a plurality of core control signals;

in said first processing mode, decoding X-bit program instruction words from a first permanent instruction set to generate said core control signals to trigger processing utilizing said N-bit data pathways; and

in said second processing mode, decoding Y-bit program instruction words from a second permanent instruction set to generate

said core control signals to trigger processing utilizing said N-bit data pathways, Y being less than X.

12. Apparatus for processing data substantially as hereinbefore described with reference to Figures 1 and 3 to 9 of the accompanying drawings.

13. A method of processing data substantially as hereinbefore described with reference to Figures 1 and 3 to 9 of the accompanying drawings.

19

Patents Act 1977 **Examiner's report to the Comptroller under Section 17** **(The Search report)**	Application number GB 9408765.7
Relevant Technical Fields (i) UK Cl (Ed.M) G4A APX, APM (ii) Int Cl (Ed.5) G06F 9/30	Search Examiner MR S J PROBERT
	Date of completion of Search 24 JUNE 1994
Databases (see below) (i) UK Patent Office collections of GB, EP, WO and US patent specifications. (ii) ONLINE: WPI	Documents considered relevant following a search in respect of Claims :- 1-13

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family; corresponding document.

Category	Identity of document and relevant passages	Relevant to claim(s)
A	US 4274138 (SHIMOKAWA) see whole document	1, 11

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal (Patents). The on-line databases considered for search are also listed periodically in the Official Journal (Patents).